

# **Creating a Winforms application**

## Table of Contents

<b>Overview</b>	<b>1</b>
<b>Prerequisites and goals</b>	<b>2</b>
<b>Creating the application</b>	<b>3</b>
<b>Understanding the WinForms Project</b>	<b>4</b>
<b>How models are constructed</b>	<b>8</b>
<b>ECO data binding</b>	<b>9</b>
ECO handles	9
<b>Creating and modifying objects</b>	<b>14</b>
<b>Running the application for the first time</b>	<b>16</b>
<b>Creating a secondary form for editing articles</b>	<b>17</b>
<b>Adding more controls to the articles form</b>	<b>21</b>
Limiting input lengths	22
<b>Observing state machine behavior</b>	<b>24</b>
Rejecting and Editing articles	26
<b>Creating a custom column</b>	<b>29</b>
<b>Using multiple EcoSpace instances</b>	<b>30</b>
<b>Associating objects using a ComboBox</b>	<b>33</b>
<b>Master detail example</b>	<b>37</b>
<b>Validating input</b>	<b>40</b>

<b>Setting maximum input lengths</b>	<b>46</b>
<b>Model driven drag 'n' drop</b>	<b>48</b>
<b>Summary</b>	<b>52</b>
<b>Index</b>	<b>a</b>

# 1 Overview

WinForms is one of the user interfaces natively supported by ECO. In a WinForms application it is possible to connect standard data binding aware user interface controls to ECO as if it were a simple dataset retrieved from a database. One of the obvious benefits over using a dataset is that your user interface will be interacting with instances of objects instead of flat data, this allows you to react to user changes via code in properties and methods.

ECO uses a set of components which act as mediators between the data binding interfaces and instances of your modeled business classes. This makes it simple to retrieve object instances using simple OCL expressions, by following associations from object instances, or to create new instances, all without writing a single line of code. These components are able to access full model information and are therefore able to provide specialized designers for performing tasks such as entering valid OCL expressions.

## 2 Prerequisites and goals

### Prerequisites

To successfully follow this document the user should have read the preceding articles in this series and have a copy of the model created.

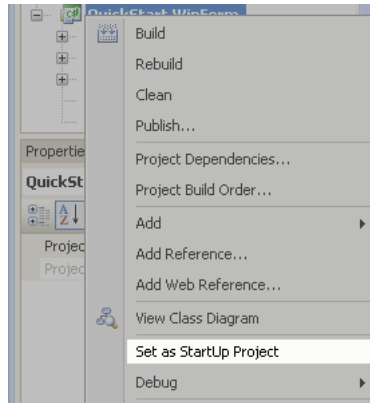
### Goals

By the end of this document you will be able to

- Create ECO WinForms applications
- Use data binding to connect standard WinForm controls to instances of modeled classes.

## 3 Creating the application

1. Open the project that you have worked with in the previous quickstart.
2. We have previously worked with QuickStart.Model, next we will work with QuickStart.WinForms. Right-click on "QuickStart.WinForms" in the solution explorer and select "Set as StartUp Project".



## 4 Understanding the WinForms Project

The WinForm project references both the QuickStart.Model project and the QuickStart.EcoSpace project. The QuickStart.EcoSpace project is referenced in order to create instances of the EcoSpace that has been defined, providing the various services for your application. The QuickStart.Model project is referenced in order to give the application access to the business classes themselves.

### ECO changes to the Program.cs file

Double-click the Program.cs file and you will see a number of changes compared with what you would normally see for a non-ECO application.

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    WinFormDequeueer.Active = true;
    Form mainForm = null;
    try
    {
        ecoSpace = new QuickStart.QuickStartEcoSpace();
        ecoSpace.Active = true;
        mainForm = new Form1(ecoSpace);
    }
    catch (Exception e)
    {
        new ThreadExceptionDialog(e).ShowDialog();
        throw;
    }
    Application.Run(mainForm);
}
```

The first line of interest is the one that reads "WinFormDequeueer.Active = true". This is described in the next section "WinFormDequeueer". A new instance of the QuickStartEcoSpace is created and passed to the main form of the application. The application is then started. This EcoSpace instance is known as the "Global EcoSpace" and is always accessible from anywhere within the application via the static Application.EcoSpace property. This instance is created for convenience only, it is possible to create new instances of the EcoSpace anywhere within your WinForms application if you wish to perform isolated operations on instances of your modeled business classes.

### WinFormDequeueer

In order to be as efficient as possible when fetching data from the data storage ECO implements a delayed fetch strategy. If for example you have a form that shows a list of customers it would be a waste of resources to retrieve thousands of objects if the user eventually looked at the first 50 before closing the form. On the other hand it would be very inefficient to fetch each customer's details individually the first time the grid attempted to read the customer's information.

The WinFormDequeueer is responsible for handling such situations. When a GUI component such as a grid displays a list of objects ECO will not immediately fetch that information. Instead all data requests are fed into a queue. When the Idle event of the application is executed the WinFormDequeueer will process the queue and retrieve the required objects in as few persistence requests as possible. For example, if you had an abstract class named "Action" and two concrete classes descended from this class named "DBAction" and "FileAction", when a grid requests a list of instances of the Action class

ECO will queue a list of objects required and then make a persistence request for each type of class encountered; in this example a selection of DBAction and a second selection of FileAction.

## Form1

Although Form1 descends from the standard System.Windows.Forms.Form it is known as an EcoForm because it implements a very simple pattern.

```
public partial class Form1 : System.Windows.Forms.Form
{
    public Form1(QuickStart.QuickStartEcoSpace ecoSpace)
    {
        InitializeComponent();

        // Set EcoSpace (actually stores it in rhRoot.EcoSpace)
        // Note that you need to set the EcoSpace property for each ReferenceHandle you
        add to the form.
        rhRoot.EcoSpace = ecoSpace;
    }

    public QuickStart.QuickStartEcoSpace EcoSpace
    {
        get { return (QuickStart.QuickStartEcoSpace)rhRoot.EcoSpace; }
    }
}
```



The constructor has been modified to accept an instance of the application's EcoSpace type. The reference to this EcoSpace instance is stored in rhRoot.EcoSpace (covered next). In addition the Form has a property named EcoSpace exposing the EcoSpace instance passed into the constructor.

This pattern is used when you wish your EcoSpace instance to be passed to the form. In this case the instance is being passed from the Application class in the Program.cs file. It is important to note that this is not a mandatory pattern. It is perfectly acceptable, and quite a common practise, to have your form create its own instance of the application's EcoSpace so that it may operate on instances of your modeled business classes in an isolated environment.

On the form there are a number of components. The only required component for integrating instances of your modeled business classes with GUI is the one named rhRoot, which is a ReferenceHandle. The standard components are:

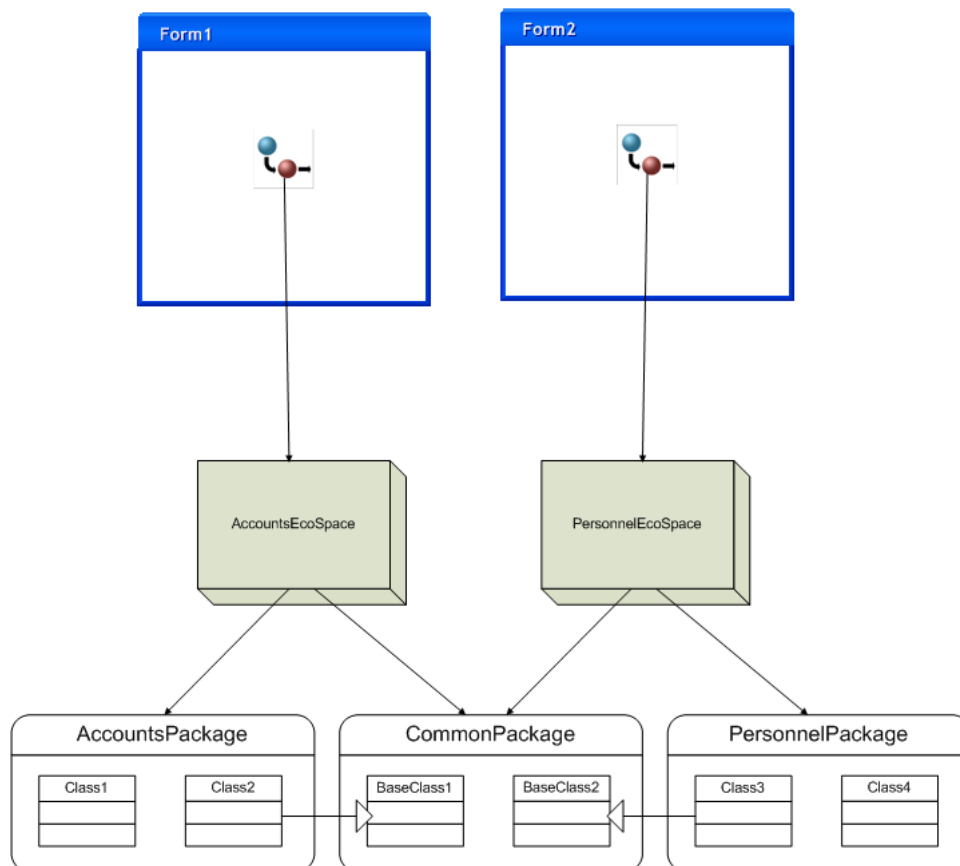


Component	Description
 rhRoot	<p><b>Design time</b></p> <p>Via its <code>EcoSpaceTypeName</code> property this component provides a link to model information, enabling the IDE integrated ECO designers to perform operations within the context of your current model.</p> <p>If this component will also hold a reference to an element (instance of a modeled business class, a simple .NET value type, or a collection of elements) then it is also possible to define a list of columns for this handle. A column being similar to a column in a grid, it is a value exposed to the .NET data binding framework. By default all properties of the element will be exposed.</p> <p><b>Run time</b></p> <p>Holds an instance of an <code>EcoSpace</code>. As this component is a "RootHandle" it is able to pass this <code>EcoSpace</code> instance onto other types of ECO components known as "handles", these are described in the next section of this article.</p> <p>As a minimum requirement this component provides its <code>EcoSpace</code> instance to other ECO handle components. If in addition its <code>Element</code> property is set this will be used as the context to any "Rooted" handles which rely on this handle for their <code>EcoSpace</code>. These child handles will be able to evaluate OCL expressions start with the keyword "self", such as "self.Articles" or "self.Articles.Comments".</p>
 EcoGlobalActions	<p><b>Run time</b></p> <p>Provides a number of actions such as</p> <ul style="list-style-type: none"> <li>• Create data storage schema</li> <li>• Evolve data storage schema</li> <li>• Activate / deactivate the <code>EcoSpace</code></li> <li>• Set an undo checkpoint</li> <li>• Undo / Redo changes</li> <li>• Commit changes to the data storage (Update Database)</li> <li>• Show <code>EcoSpace</code> debugger</li> <li>• Enabling / disabling controls based on evaluating an OCL expression</li> </ul> <p>The actions are invoked from standard WinForm controls such as <code>Button</code>. With an <code>EcoGlobalActions</code> component on your form these standard controls will have additional ECO related properties allowing you to perform the above actions directly from the user interface. The <code>EcoSpace</code> instance to perform these operations on is obtained via the components "RootHandle" property which by default is connected to the "rhRoot" component.</p>
 EcoListActions	<p><b>Run time</b></p> <p>Provides a number of actions such as</p> <ul style="list-style-type: none"> <li>• Adding new object instances</li> <li>• Deleting object instances</li> <li>• Unlinking an object instance from an association - <code>AnObject.SomeAssociation.Remove(SelectedObject)</code></li> <li>• Move object up/down in an association.</li> </ul> <p>There are many more, you can see the list by inspecting the <code>EcoListAction</code> property on a <code>Button</code> for example.</p>

 EcoDragDrop	<b>Run time</b> <p>Expands controls such as DataGridView to enable automatic drag and drop operations. This is a good example of one of the benefits of having full model information available at runtime.</p> <p>This component allows you to identify whether or not a control is a drag source, a drop target, or both. Your user will then be able to associate object instances with each other simply by using standard Windows drag/drop operations.</p> <p>For example, the user could drag a Person from list in a DataGridView and drop it onto another DataGridView showing a list of residents of a building. This is possible because ECO uses model information to determine that an instance of Person is compatible with the Building.Residents displayed in the target control.</p>
 EcoAutoForms	<b>Run time</b> <p>Extends standard controls such as the DataGridView so that double-clicking an object will display an auto-generated form for that object. The form displayed is the same as the kind you saw in the Prototyping article in this series.</p> <p>It is recommended that auto-generated forms are used only for testing purposes due to the fact that they allow the user complete access to your object instance and this may be undesirable.</p> <p>It is possible to plug into this mechanism in order to provide your own pre-defined form rather than an auto-generated one. These are known as "auto forms" as opposed to "auto generated forms". An example of such a mechanism may be found in the ECO Extensions, which come with documentation.</p>

## 5 How models are constructed

A single application may contain more than one type of EcoSpace. To define the model for an EcoSpace the developer must select one or more UML Packages. UML Packages may be used in more than one EcoSpace type, and in more than one application. In the following diagram both EcoSpace types use the CommonPackage, in addition to this both the AccountsPackage and the PersonnelPackage have classes that descend from classes within the CommonPackage.

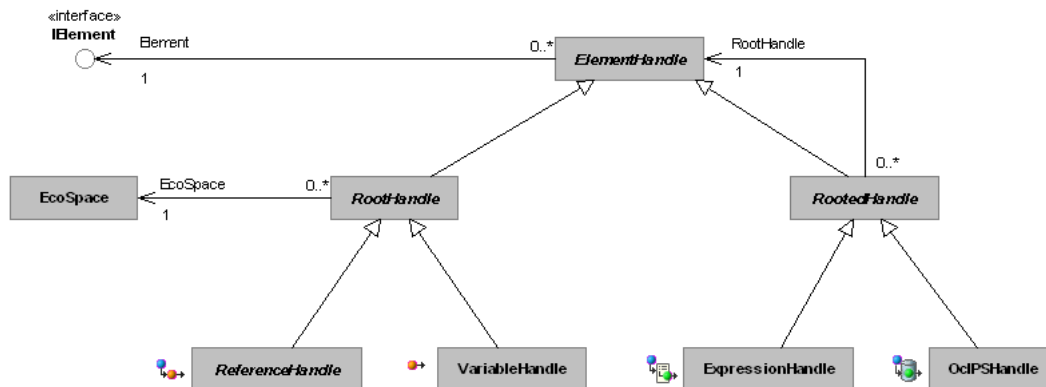


As you can see ECO will allow for much more sophisticated scenarios than the single-package single EcoSpace approach demonstrated in these QuickStart articles.

## 6 ECO data binding

ECO "handle" components are either RootHandle descendants or RootedHandle descendants. A RootHandle holds a direct reference to an EcoSpace instance via its EcoSpace property, and optionally holds a direct reference to an element via its Element property.

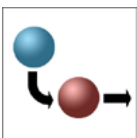
A RootedHandle always connects to another handle. Its context will be the element of the handle it is connected to, its EcoSpace will be obtained ultimately from the RootHandle.



This section describes some of the available ECO handles in detail. Some of these handles will be demonstrated as this article progresses, this section may be used as a reference to help understand the steps as you follow the steps outlined.

### 6.1 ECO handles

#### ReferenceHandle



The ReferenceHandle component is the most commonly used of all the ECO handles. At the very least this handle will hold a reference to the EcoSpace at runtime, and a link to the type of the application's EcoSpace at design time in order to provide the ECO designers with model information. Optionally this handle may also hold a reference to a specific element such as an instance of the Author class.

#### ExpressionHandle



The ExpressionHandle is a RootedHandle descendant, therefore it depends on another handle component to obtain a context and a reference to an EcoSpace. The ExpressionHandle evaluates a specific OCL expression in order to determine

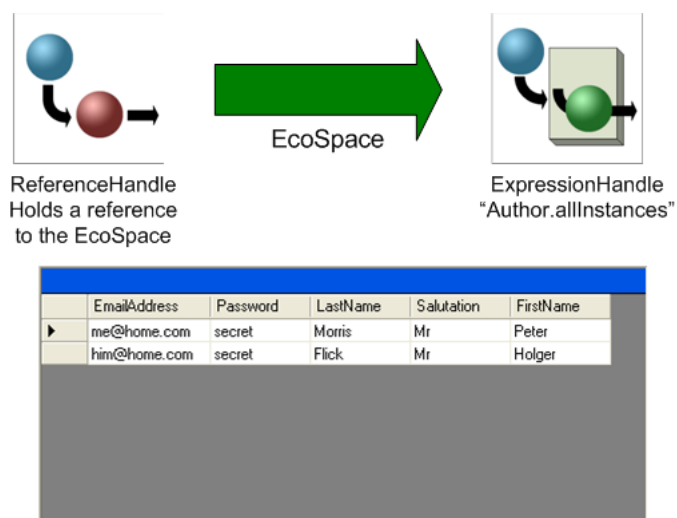
its Element. For example, if the RootHandle's Element returns an instance of the Author class then the following expression would return a list of the author's articles.

```
self.Articles
```

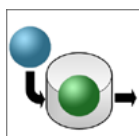
whereas if the RootHandle's Element returns nil/null (no context) then only model based expressions are valid

```
Article.allInstances
```

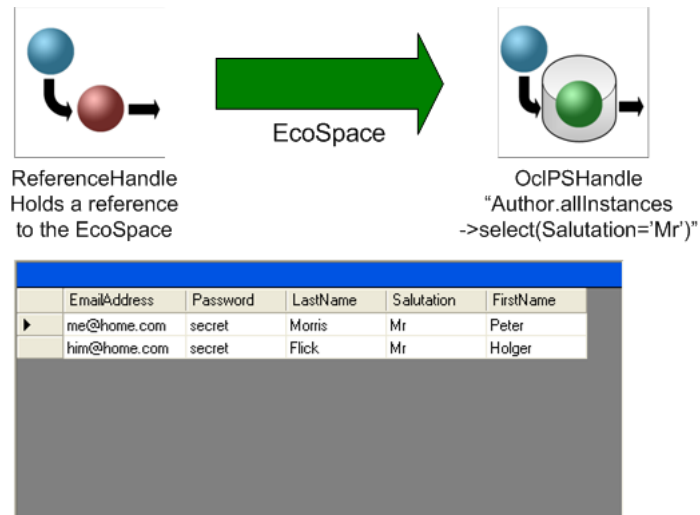
would return all Articles in the data store.



### OciPSHandle



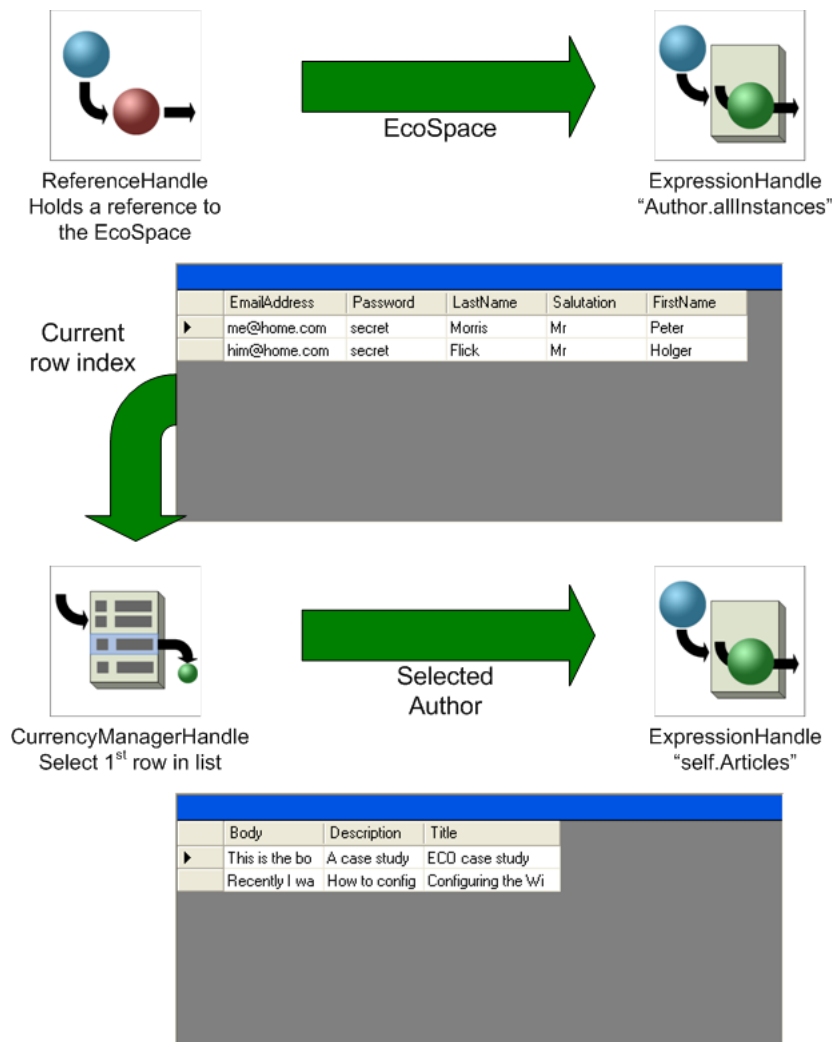
The OciPSHandle performs the same task as the ExpressionHandle but instead of evaluating the OCL expression in memory it will transform the expression into SQL and execute it in the database server. If the data store is not a SQL database then evaluation will be performed in memory. The developer must explicitly invoke the OciPSHandle's "Execute" method in order to evaluate its result.



### CurrencyManagerHandle



The **CurrencyManagerHandle** may be used to implement a typical Parent / Child interface. This component links to a **RootHandle** for its context, and a WinForms control such as a **DataGrid** via its **BindingContext** property. Whenever the user changes the selected row of the WinForms control the **CurrencyManagerHandle**'s element will also change to reflect the object selected.



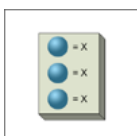
If you wish to link a single value control such as a TextBox to an ExpressionHandle then a CurrencyManagerHandle is not needed. When your parent handle holds a collection of elements and you want a child handle to use only a single element from the collection as its context then a CurrencyManagerHandle is needed, otherwise the child handle will use the collection as its context.

For example, the OCL

```
self.Articles
```

evaluated against the context of a single Author will return the articles belonging to the individual author, whereas if it is evaluated against a collection of authors it will return all of the articles belonging to every author in the collection.

### OclVariables



Although the OclVariables component is not an ECO handle, it does provide support for the various ECO handles so is listed

here for the sake of completeness. The `OclVariables` component allows the developer to specify a collection of named elements, any ECO handle that connects to this component via its "Variables" list has access to these elements via their names. These variables are then available in all OCL expressions, so an `ExpressionHandle` that uses an `OclVariables` component would be able to specify OCL similar to the following:

```
Article.allInstances->select(rating >= UserEnteredRatingVariable)
```

This is useful when you want to affect the resulting of OCL evaluations on handles such as the `ExpressionHandle` without having to write code to alter its "Expression" property.



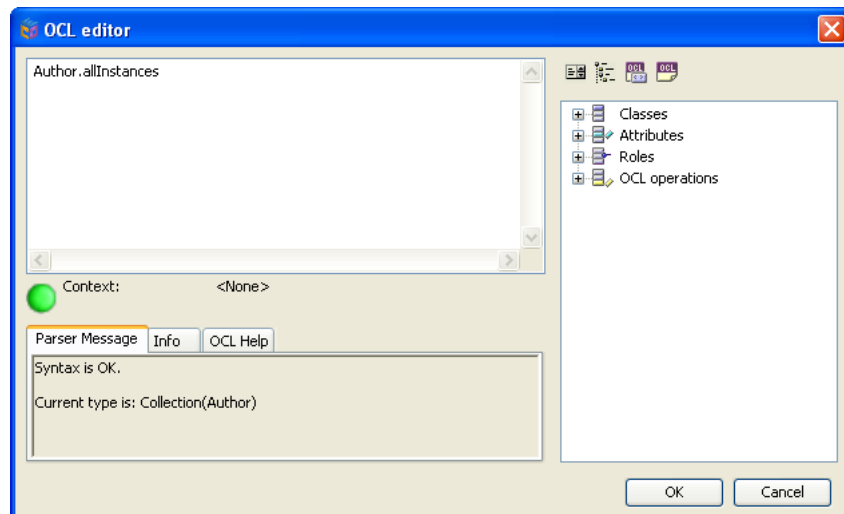
## 7 Creating and modifying objects

1. Add a new ExpressionHandle to the form.
2. Rename the ExpressionHandle to "ehAuthors".
3. Ensure that its RootHandle property is set to rhRoot. This property has two purposes:
  1. At design time the ExpressionHandle can find the EcoSpaceTypeName and therefore obtain model information.
  2. At runtime the ExpressionHandle can obtain a reference to the EcoSpace instance in order to create instances of modeled business classes, delete instances, or modify their property (column) values.
4. Click the [...] editor on the expression handle's "Expression" property. ECO will now use the model information from the specified EcoSpaceTypeName to present you with a model aware OCL editor. Ensure you have done a Build on the solution first to ensure the binaries are up to date.
5. In the editor enter the expression

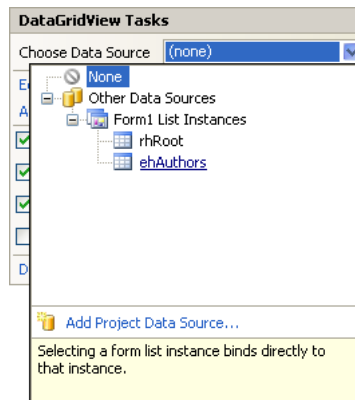
```
Author.allInstances
```

7

6. The OCL editor will confirm that the expression is valid, and at the bottom indicate that the result of the expression is "Collection(Author)".



7. Click the "OK" button.
8. Now that ehAuthors has a context (a collection of Author instances) right-click the component and select "Create default columns".
9. Set the property AddDefaultProperties to False. At runtime a Column will be added for each property on the class, as we want full control over which columns are created this should be set to False.
10. Click the [...] editor on the Columns property and remove Articles, FullName, and ID.
11. Add a DataGridView to the form.
12. Name it "GridViewAuthors".
13. Set its DataSource property to ehAuthors.



14. Add a button to the form.

15. Name it "ButtonAddAuthor" and give it suitable text.

16. In order to add a new author you have two options.

Option 1 - Using the Click event on the button

```
private void ButtonAddAuthor_Click(object sender, EventArgs e)
{
    new Author(EcoSpace);
}
```

Option 2 - Using the EcoListActionExtender component

1. Set the "RootHandle on EcoListActions" property to "ehAuthors".

2. Set the "EcoListAction on EcoListActions" property to "Add".

Note: Doing both will result in two new Authors being created each time the user clicks the button.

## 8 Running the application for the first time

1. Run the application.
2. Enter a few rows of data.

EmailAddress	FirstName	LastName	Password	Salutation	AutoPublishArticles
me@home.com	John	Smith	secret	Mr	<input checked="" type="checkbox"/>
him@home.com	Fred	Smith	secret	Mr	<input type="checkbox"/>

Add author

3. Close the application.
4. You will receive an exception: "Deactivating system with dirty objects." because there were modified objects that weren't saved when you closed the application.  
You can avoid this exception by setting the property "AllowDeactivateDirty" on the EcoSpace (in the QuickStart.EcoSpace project)

5. Stop the debugger.
6. Now restart the application and you will notice that the data you previously entered has now disappeared.  
This is because when you add or change a row you are modifying the local cache held within the EcoSpace, no data storage updates have been performed. This is similar to the way a database transaction would work where you must explicitly commit the changes that occurred within it.

5. Add another button and name it "ButtonUpdateDB" and give it suitable text.
6. Add the following code to the button:

```
private void ButtonUpdateDB_Click(object sender, EventArgs e)
{
    EcoSpace.UpdateDatabase();
}
```

or set the "EcoAction on EcoGlobalActions" property on the button to UpdateDatabase.

Now run the application again, enter two rows of data, but before closing the form click the Update DB button. Close the application and then rerun it again, this time the data has survived beyond the life of the application.

NOTE: The data has been persisted into a file named Data.xml in the application's folder.

## 9 Creating a secondary form for editing articles

Next a secondary form will be created. This form will accept an instance of the Author class and display their articles for editing.

1. Right click the QuickStart.WinForms node in the solution explorer.
2. From the context menu select Add->Windows Form.
3. Name the item "AuthorArticlesForm.cs" and click "Add".

Note: Although it is possible to add an ECO WinForm template this article will perform the steps manually to try to help understand how the template works.

4. Add a ReferenceHandle component to the form and name it rhAuthor.
5. Set its EcoSpaceTypeName property from the drop down list.
6. Set its StaticValueTypeName property to "Author" from the drop down list. This informs the ECO designers that at runtime the rhAuthor.Element will hold a reference to an instance of the Author class or an instance of a descendant class.

Now we have design time support, but the reference handle must obtain an EcoSpace from somewhere at runtime.

7. Change the constructor for the form so that it accepts an EcoSpace instance.

```
public AuthorArticlesForm(QuickStartEcoSpace ecoSpace, Author selectedAuthor)
{
    if (ecoSpace == null)
        throw new ArgumentNullException("ecoSpace");
    if (selectedAuthor == null)
        throw new ArgumentNullException("selectedAuthor");

    InitializeComponent();
    rhAuthor.EcoSpace = ecoSpace;
    rhAuthor.SetElement(selectedAuthor);
}
```

For good practise the constructor ensures we are not passed a null reference. Although this would only result in an inactive GUI at runtime it is better for the developer to see the exception during testing, the bug is easier to identify this way. Then the EcoSpace instance passed is stored in the EcoSpace property of rhAuthor. Finally the author instance itself is stored in rhAuthor.Element by executing rhAuthor.SetElement.

8. Next add a property to the form exposing the EcoSpace. This is not a mandatory step but does help when writing ECO code within the form itself. This property type casts the rhAuthor.EcoSpace property to the EcoSpace type of the current project. This is useful if you have defined additional methods or properties on the EcoSpace class.

```
public QuickStartEcoSpace EcoSpace
{
    get { return (QuickStartEcoSpace)rhAuthor.EcoSpace; }
}
```

We now need a way to invoke this form from the main form for the selected author.

9. Switch your designer back to Form1 and add a new button to the form.
10. Name the button ButtonShowArticles and set its caption "Show Articles".
11. In its Click event add the following code:

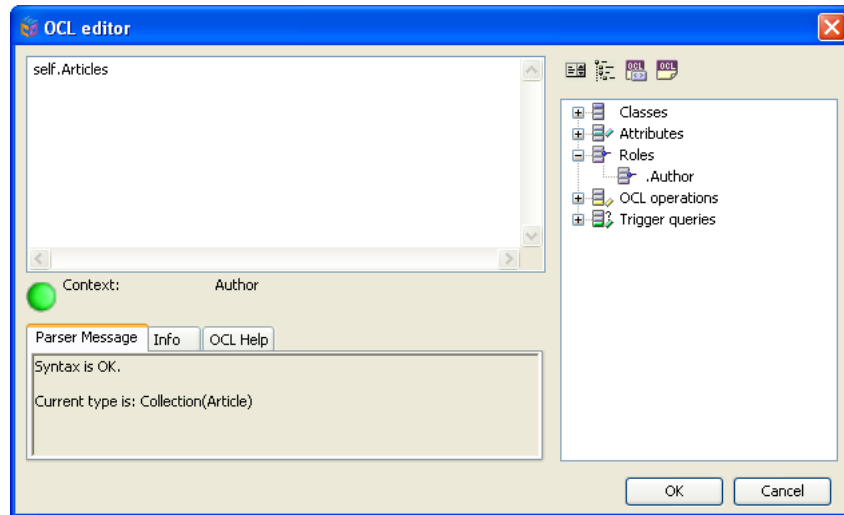
```
private void ButtonShowArticles_Click(object sender, EventArgs e)
{
    IElement selectedAuthorElement =
    CurrencyManagerHandle.CurrentElement(GridViewAuthors);

    if (selectedAuthorElement != null)
    {
        Author selectedAuthor = selectedAuthorElement.GetValue<Author>();
        AuthorArticlesForm authorArticlesForm = new AuthorArticlesForm(EcoSpace,
        selectedAuthor);
        authorArticlesForm.ShowDialog();
    }
}
```

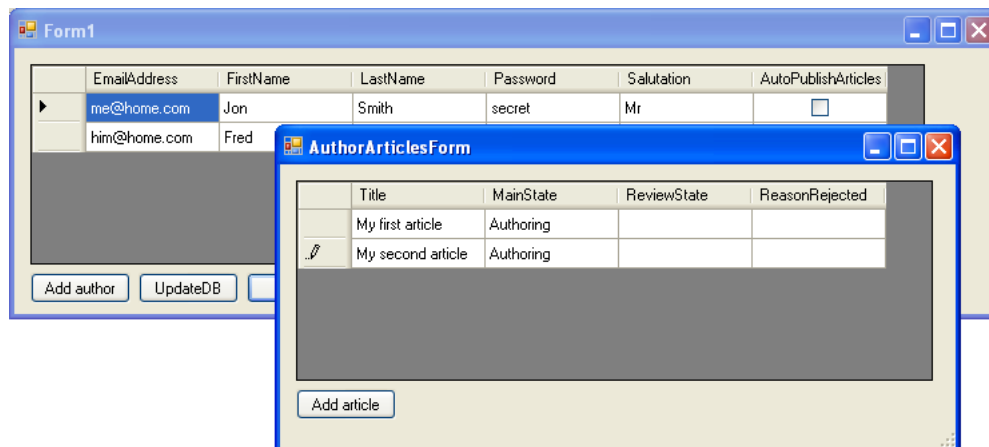
- This code uses a static method on the CurrencyManagerHandle class to get the currently selected object in the specified control "GridViewAuthors".
- The result of the CurrentElement method is an IElement, which is an ECO interface used to identify cached values such as object instances and even simple data types.
- The selected author instance is obtained using selectedAuthorElement.GetValue<Author>().
- Finally an instance of the form is created and the EcoSpace + selected author are passed.

Next some GUI will be added to the articles form.

13. Switch back to the designer for the AuthorArticlesForm.
14. Add an ExpressionHandle to the form and name it "ehAuthorArticles".
15. Set its RootHandle property to point to rhAuthor, this will provide it with an EcoSpace instance and an Author for the context of any OCL expression evaluation.
16. Set its expression to self.Articles - this will confirm the expression is valid and indicate the return type will be a collection of Article.



21. Set AddDefaultProperties to False.
22. Right-click the component and select "Create default columns".
23. Click the [...] property editor on the Columns property, remove the ID and Author columns.
24. Ensure that the column for ReasonRejected is set to be read-only.
25. Add a DataGridView to the form and name it GridAuthorArticles.
26. Set its DataSource to ehAuthorArticles.
27. You may optionally edit the columns of GridAuthorArticles to show only Title, MainState, ReviewState, and ReasonRejected.
28. Set ReasonRejected's ReadOnly = True.
29. Add an EcoListActionExtender component to the form and name it "EcoListActions".
30. Add a button to the form, name it "ButtonAddArticle" and give it suitable text.
31. Set its "RootHandle on EcoListActions" to "ehAuthorArticles" and "EcoListAction on EcoListActions" to "Add".



Now run the application. Select an author you entered earlier and click the "Show articles" button and add an article or two for this author.

Because this form uses the same EcoSpace as the main form it is necessary to click the Updated DB button on the main form if you wish to commit your changes to the data store.

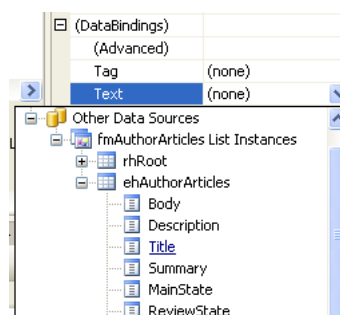
**Exercise for the reader**

Create another form for editing instances of the modeled class `ArticleType` and add a button to show the form from the main form. Note that these instances will be required later in this article!

# 10 Adding more controls to the articles form

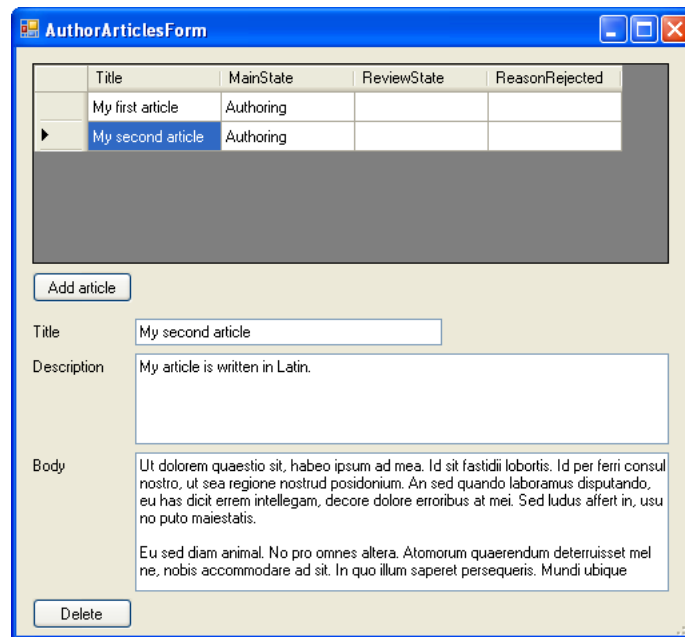
Next some more controls will be added to the articles form to allow the user to enter more information about the article.

1. Add a TextBox to the form.
2. Name it "TextBoxTitle".
3. Under "(Databindings)" bind the "Text" property to "ehAuthorArticles - Title".



4. Add another TextBox named "TextBoxDescription" and data bind its Text property to "ehAuthorArticles - Description".
5. Set its Multiline property to "True".
6. Add yet another TextBox named "TextBoxBody" and data bind its Text property to "ehAuthorArticles - Body".
7. Set its Multiline property to "True".
8. Add a button named "ButtonDelete".
9. Set its Text property to "Delete".
10. Set its "EcoListAction on EcoListActions" to "Delete".
11. Set its "RootHandle on EcoListActions" to "ehAuthorArticles".
12. Set its BindingContext to "GridAuthorArticles"





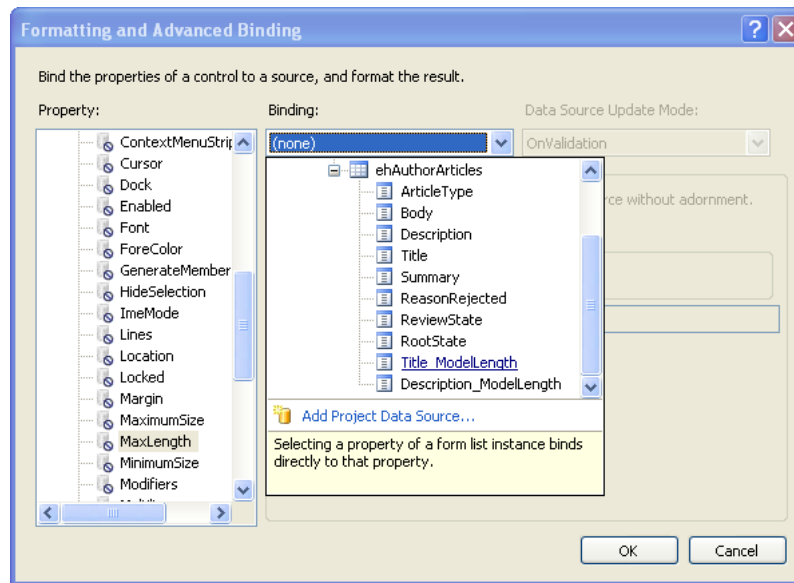
## 10.1 Limiting input lengths

The GUI created so far will allow the user to enter strings of any length. It is possible in WinForm applications to set the maximum input length for controls in order to prevent the entered value from being truncated when saved to a data storage. Instead of looking at the sizes in the model and then setting these manually it is possible to obtain the model information via an OCL expression and data bind to the result. This has the obvious benefit that the value is determined by the model, so changing the length of a UML attribute requires a change in a single place only, all GUI will then data bind to that new value.

1. Click the Columns [...] editor on ehAuthorArticles.
2. On the editor form click the "Add" button.
3. A new column will be added, give it a name that clearly indicates it is not part of the class, for example "Title\_ModelLength".
4. Set its OCL expression to

```
self.Title.MaxLength
```

5. Repeat this procedure for Description, but not for Body as it is capable of accepting an unlimited length.
6. Click TextBoxTitle and expand its DataBindings property.
7. Click the Advanced [...] editor within its sub properties.
8. In the list on the left hand side select "MaxLength".
9. Finally, from the "Binding" drop down list select "ehAuthorArticles - Title\_ModelLength".



Repeat this procedure for `TextBoxDescription` and run the application. You will now observe that it is not possible to enter more than 64 characters for the Title of an article.

# 11 Observing state machine behavior

Next the user will be able to submit their article for publication. We will observe how the object behaves differently depending on the value of `Author.AutoPublishArticles`.

1. Add a Button to the form `AuthorArticlesForm`.
2. Name it "ButtonPublish".
3. In the `OnClick` event enter the following source code:

```
private void ButtonPublish_Click(object sender, EventArgs e)
{
    if (CurrencyManagerHandle.CurrentElement(GridAuthorArticles) != null)
    {
        Article selectedArticle =
        CurrencyManagerHandle.CurrentElement(GridAuthorArticles).GetValue<Article>();
        try
        {
            selectedArticle.Publish();
        }
        catch (NoTransitionException E)
        {
            MessageBox.Show("Cannot publish an article in this state!");
        }
    }
}
```

Note: You will need `Eco.Handles` and `Eco.ObjectRepresentation` in your file's using clause.

4. Run the application.
5. Ensure that there is one Author with `AutoPublishArticles` set to `True`. Select this author and click "Show articles".
6. Select an existing article, or create a new article and save it.
7. Click the Publish button. You will now observe the `MainState` has changed to `Published`.

	Title	MainState	ReviewState	ReasonRejected
▶	Mr first article	Published		
	My second article	Authoring		

Title:

Description:

Body:

8. Close the articles form.
9. Select the Author with AutoPublishArticles set to False and click "Show articles".
10. Again ensure you have an article for this author and click the Publish button.
11. This time you will see that the MainState has changed to Reviewing and the ReviewState has changed to CheckingSpelling.

	Title	MainState	ReviewState	ReasonRejected
▶	My first article	Reviewing	CheckingSpelling	
	My second article	Authoring		

Title:

Description:

Body:

## 11.1 Rejecting and Editing articles

Next the GUI will be updated to allow the user to control the state of an article in full.

1. Delete the Click event code for "ButtonPublish".
2. Select "ButtonPublish" and set its "RootHandle on EcoListActions" to "ehAuthorArticles".
3. Set its "EnabledOcl on EcoListActions" to the following OCL expression

```
self.Publish?
```

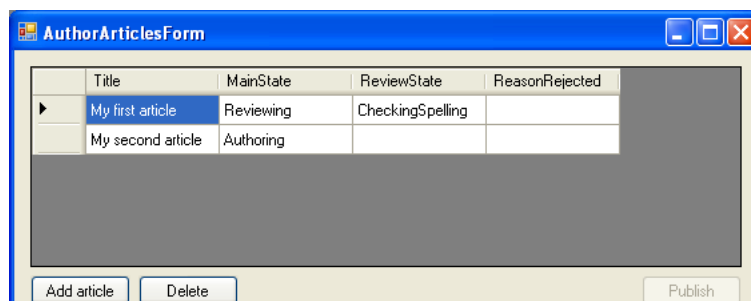
Note: "self" is case sensitive and should be in lower case.

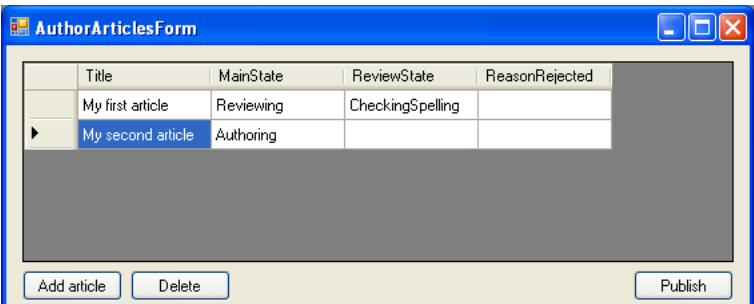
This query checks if it is possible to execute the Publish trigger on the article. If the current state of the article does not have a transition with this trigger the result will be False, otherwise the expression will evaluate as True. This will ensure that ButtonPublish is only enabled if it is possible to publish the selected article. To execute the Publish trigger you can either use a Click event with code similar to how it was before, or you can use the ECO OCL Action language.

4. Set the button's "EcoListAction on EcoListActions" property to "ExecuteAction".
5. Set the "ActionExpression on EcoListActions" for ButtonPublish to the following OCL expression

```
self.Publish
```

6. Run the application again.
7. As you select articles with different states you will observe the Publish button alternating between Enabled and Disabled.
8. When you click the Publish button in its enabled state you will note that the Publish trigger is executed on the select article's state machine, and the article enters either the Published or Reviewing state.





Now repeat these steps for the following triggers:

ButtonSpellingOkay

Property	Value
RootHandle on EcoListActions	ehAuthorArticles
EnabledOcl on EcoListActions	self.SpellingChecked?
EcoListAction on EcoListActions	ExecuteAction
ActionExpression on EcoListActions	self.SpellingChecked

ButtonGrammarOkay

Property	Value
RootHandle on EcoListActions	ehAuthorArticles
EnabledOcl on EcoListActions	self.GrammarChecked?
EcoListAction on EcoListActions	ExecuteAction
ActionExpression on EcoListActions	self.GrammarChecked

ButtonReject

Property	Value
RootHandle on EcoListActions	ehAuthorArticles
EnabledOcl on EcoListActions	self.Reject?
EcoListAction on EcoListActions	ExecuteAction
ActionExpression on EcoListActions	self.Reject

ButtonEdit

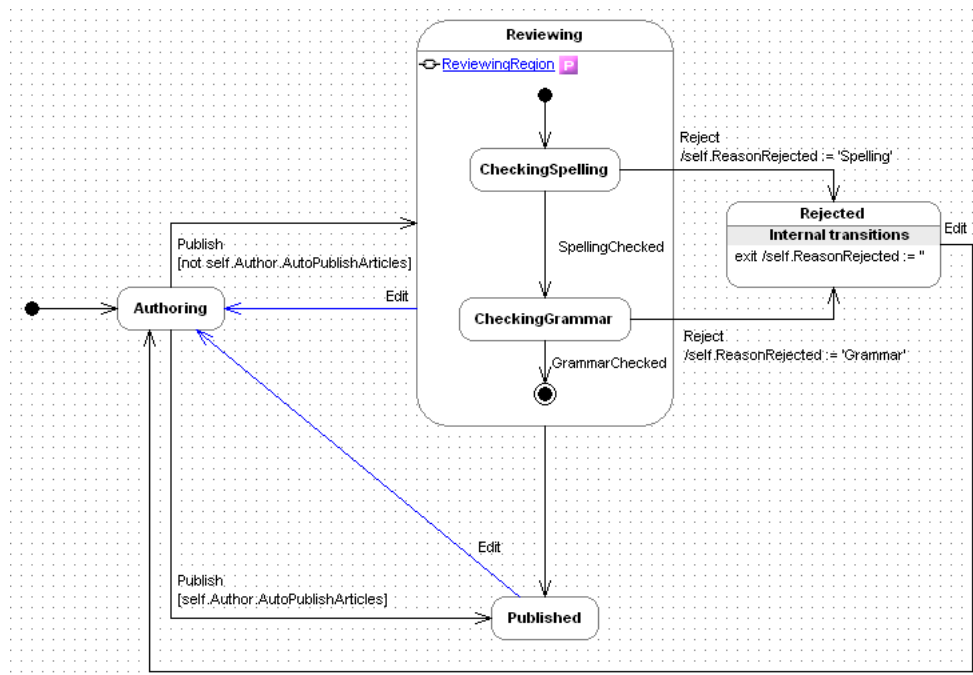
Property	Value
RootHandle on EcoListActions	ehAuthorArticles
EnabledOcl on EcoListActions	self.Edit?
EcoListAction on EcoListActions	ExecuteAction
ActionExpression on EcoListActions	self.Edit

Now run the application again. As you select different articles the buttons enable or disable automatically. As you execute the triggers by clicking the buttons you will see the states update in the grid and the buttons enable or disable too. Note also that when you Reject an article the ReasonRejected will display the reason why, and when you click Edit the reason is cleared.

One of the benefits of this approach is that you do not need to enable or disable buttons in the UI based on the current state. If you were to take this approach you would need to remember to update your form when the state machine definition in your model changes, you may even have to change multiple forms. This approach uses the rich model information at runtime to determine whether or not a trigger is valid, a change to the model requires no changes to your GUI.

### Exercise for the reader

When the final state of the Reviewing region is reached it displays the state "Final\_1". Edit the model so that this is named "Reviewed" instead. Additionally, you may wish to add a transition back to the Authoring state from both Published and Reviewing; this would allow them to retract an article and also intervene immediately if they notice an error in their article during the reviewing stage. The user interface will handle the changes automatically.



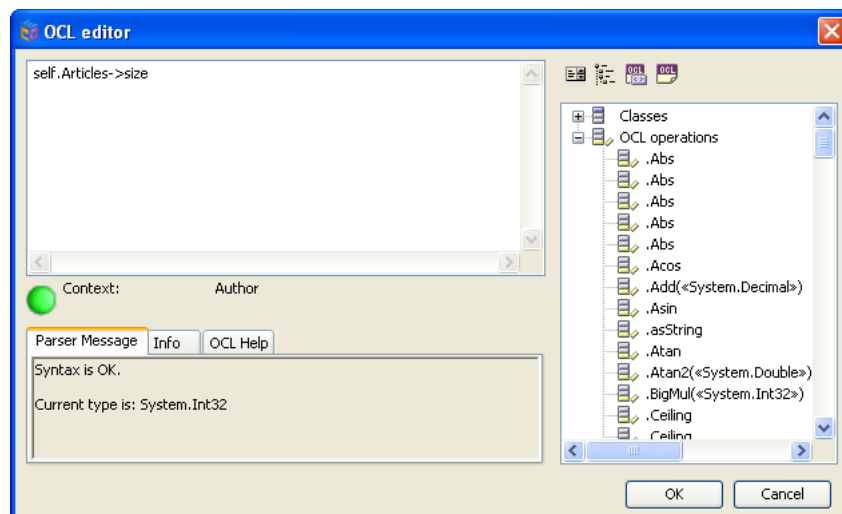
## 12 Creating a custom column

A column on an ECO handle is a point of exposure for data binding. Not only can you choose to expose fewer columns than there are members on your class, but you can also add additional columns non-modeled values too. This allows you to add interface specific data without having to modify your business model in order to satisfy user interface requirements.

As an example a custom column will be added to the Author class indicating how many articles the author has written.

1. Select the ehAuthors component on the main form.
2. Click the [...] property editor on the Columns property.
3. Add a new column named "NumberOfArticles".
4. Set IsReadOnly to True.
5. Click the [...] property editor for the Expression property.
6. In the OCL editor enter the following expression:

```
self.Articles->size
```



7. Click the "OK" button.
8. Ensure that the new NumberOfColumns column is added to the Columns property of GridViewAuthors.
9. Now run the application again.
10. Select an Author and make a note of the value in NumberOfArticles.
11. Click the "Show articles" button.
12. Click the button to add a new article.
13. You will observe NumberOfArticles updates automatically.
14. Select the new article and click the Delete button.
15. Again the NumberOfArticles will update.



## 13 Using multiple EcoSpace instances

There are various ways of isolating changes made within forms from other forms in the application, allowing the user to commit or revert their changes without affecting the data in other forms. One of these techniques is to use a new instance of the application's EcoSpace class in each form. This technique will be demonstrated next.

1. Change the constructor of AuthorArticlesForm to the following source code:

```
public AuthorArticlesForm(Author selectedAuthor)
{
    if (selectedAuthor == null)
        throw new ArgumentNullException("selectedAuthor");

    InitializeComponent();
    rhAuthor.EcoSpace = new QuickStartEcoSpace();

    rhAuthor.EcoSpace.Active = true;
    rhAuthor.SetElement(selectedAuthor);
}
```

Note: The "ecoSpace" parameter has been removed, and rhAuthor.EcoSpace is instead set to a new instance of the QuickStartEcoSpace.

2. Now change the source code in the main form that creates an instance of this form:

```
AuthorArticlesForm authorArticlesForm = new AuthorArticlesForm(selectedAuthor);
```

3. Now run the application.
4. Select an author and click "Show articles".

At this point you should experience an exception informing you that the value you are trying to use in rhRoot.SetElement is an object that belongs to another EcoSpace. The reason you cannot mix EcoSpaces is to avoid problems with associations. If ObjectA belonged to instance A of an EcoSpace and ObjectB belonged to instance B of an EcoSpace, ECO would prevent you from executing code like this:

```
ObjectA.AssociationToB := ObjectB;
```

If this association were to be allowed, and then EcoSpace instance B was deactivated, ObjectA would be holding a reference to an object that no longer has access to its owning EcoSpace. This would result in ObjectA.AssociationToB.Name := 'Hello' attempting to write to the cache of a deactivated EcoSpace, which would most likely throw a NullReferenceException.

NOTE: At the time of writing this exception is not thrown when calling SetElement, but there are plans to ensure it will be in future.

5. Change the constructor of AuthorArticlesForm again:

```
public AuthorArticlesForm(string selectedAuthorID)
{
    if (string.IsNullOrEmpty(selectedAuthorID))
        throw new ArgumentNullException("selectedAuthorID");

    InitializeComponent();
    rhAuthor.EcoSpace = new QuickStartEcoSpace();
    rhAuthor.EcoSpace.Active = true;

    Author selectedAuthor =
    EcoSpace.ExternalIds.ObjectForId(selectedAuthorID).GetValue<Author>();
    rhAuthor.SetElement(selectedAuthor);
}
```

Instead of an Author instance being passed to the constructor the main form will now pass a string identifying the object. One of the ECO services (The External ID Service) is used to load the specified object from the data storage, this is then cast to an Author object. The External ID Service allows you to retrieve a string identifier for an object, and an object instance for a string identifier. This string may be used across different EcoSpace instances as long as the object has been saved.

NOTE: SetElement will also accept the result of ObjectForId directly, there is no need to convert it to a business object (using .GetValue<Author>).

6. Next the opposite method of ObjectForId will be used in order to retrieve the ID string for the selected author, which will then be passed to the AuthorArticlesForm.

```
AuthorArticlesForm authorArticlesForm = new
AuthorArticlesForm(EcoSpace.ExternalIds.IdForObject(selectedAuthor));
```

7. Now drop a new button onto the AuthorArticlesForm.

8. Name it ButtonUpdateDatabase.

9. Either set its Click event to the following:

```
EcoSpace.UpdateDatabase();
```

or use an EcoActionExtender to attach an "UpdateDatabase" action to it.

10. Run the application again.

11. Select an Author.

12. Ensure the NumberOfArticles column is visible and make a note of its current value.

13. Click "Show articles".

14. Create a new article and Post the changes.

15. Click the Commit button.

16. Now close the form.

You might notice that this time the NumberOfArticles value has remained unchanged. This clearly shows that the changes

made in the secondary form were completely independent from the values in the main form. To refresh the NumberOfArticles you can add the following code:

```
selectedAuthor = selectedAuthorElement.GetValue<Author>();
authorArticlesForm = new
AuthorArticlesForm(EcoSpace.ExternalIds.IdForObject(selectedAuthor));
authorArticlesForm.ShowDialog();

//Add the two following lines of code
EcoSpace.Active = false;
EcoSpace.Active = true;
```

Note: This is not the "ECO way" of refreshing the data. It will suffice until multi-user synchronization is covered later in this series.

# 14 Associating objects using a ComboBox

Next a column will be added to GridAuthorArticles which will allow the user to specify an ArticleType for an article. To perform this task you will need some instances of the ArticleType class saved in your data storage.

First a list of article types is required to display in the combo box's drop down list.

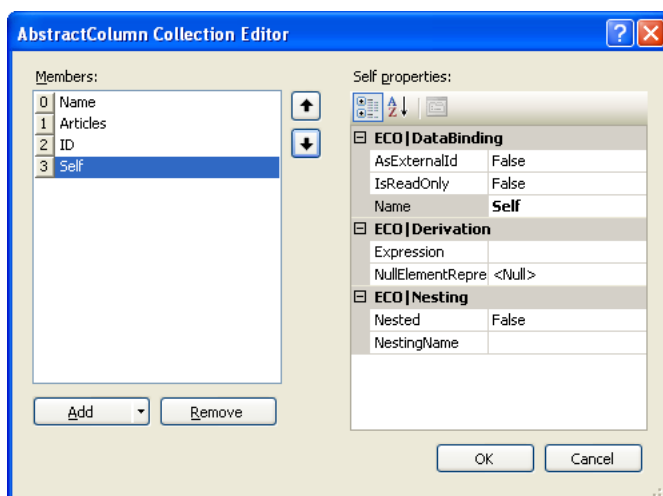
1. Add an ExpressionHandle to the form and name it "ehArticleTypes".
2. Set its RootHandle to rhAuthor.
3. Set its expression to

```
ArticleType.allInstances->orderBy(name)
```

At runtime this handle will hold a collection of ArticleType, ordered alphabetically.

Note: Even though the context for ehArticleTypes is a single Author instance it is still possible to retrieve top-level model information by starting the expression with the name of a class. This is why the keyword "self" is used at the start of many expressions, in order to avoid ambiguity. E.g. "Author" and "self.Author".

4. Right-click the handle and select Create Default Columns from the context menu.
5. Edit the columns for the handle.
6. Add a new column by clicking the "Add" button.
7. Give the new column the name "Self".
8. Leave its expression property empty.
9. Click the "OK" button.



The expression was left blank but could just have easily been set to the keyword "self". The context for the expression is an instance of the modeled ArticleType class, if no expression text is entered the result of the evaluation will be the same as the context. In this case an instance of ArticleType.

Now we have a list of all available ArticleType instances and a column identifying the object itself for each row. Next the ArticleType column on ehAuthorArticles will be modified so that it may be set to the value of the "Self" column on ehArticleTypes.

10. Select the Columns [...] editor on ehAuthorArticles.

11. Select the ArticleType column.

12. Change the expression for this column so that it reads "self.ArticleType"

Note: The expression for this column ends with .asString - this presents the object as a string by evaluating the Default String Representation modeled on that class (if you have set one, for example "name").

13. Ensure that IsReadOnly is false.

14. Click the "OK" button.

Finally a new column will be added to the grid showing the article type for each article.

15. Select the Columns [...] editor on GridAuthorArticles.

16. Click the "Add..." button.

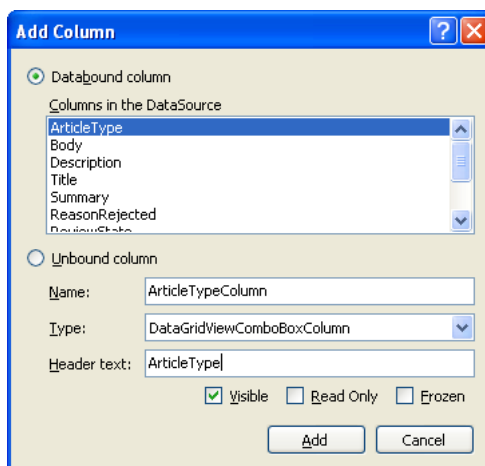
17. A new property form will appear, set the following properties

Property	Value
Name	ArticleTypeColumn
Type	DataGridViewComboBoxColumn (selected from the drop down list)
Header text	Article type

18. Ensure that the "Databound column" radio button is selected and choose "ArticleType" in the list.

19. Click the "Add" button.

20. Click the "Close" button.



21. With the ArticleType column selected scroll down the property list on the right hand side of the "Edit columns" window until you reach the "Data" section.

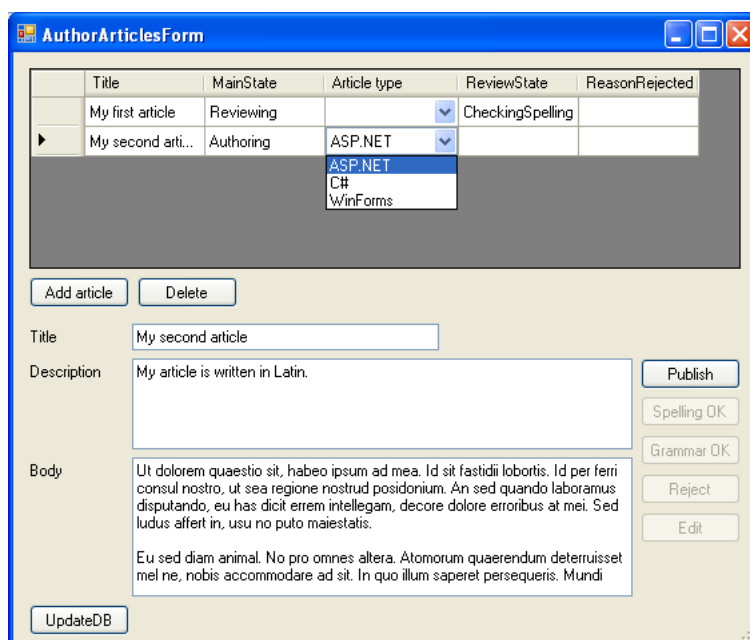
22. DataPropertyName should read "ArticleType", this is the actual value set in the Article.

23. For the DataSource select "ehArticleTypes", this identifies the source of the data items to appear in the list.

24. For DisplayMember select "Name".

25. Set the ValueMember property to the column named "Self".

When you next run the application and edit your articles you will see an additional column allowing you to specify the type of the article.



The drop down list is populated from the columns DataSource property (ehArticleTypes). Every item displayed is the value of DisplayMember (name). Whenever you change the selected index the ValueMember (a column named "Self") value is used to set the current article's ArticleType column.

If you wish to allow the user to specify no article type set ehArticleTypes.NullRowMode to "First", this will add a <null> as the first item in the collection of available ArticleType instances.

Note: To data bind to a standard combo box you would set following properties.

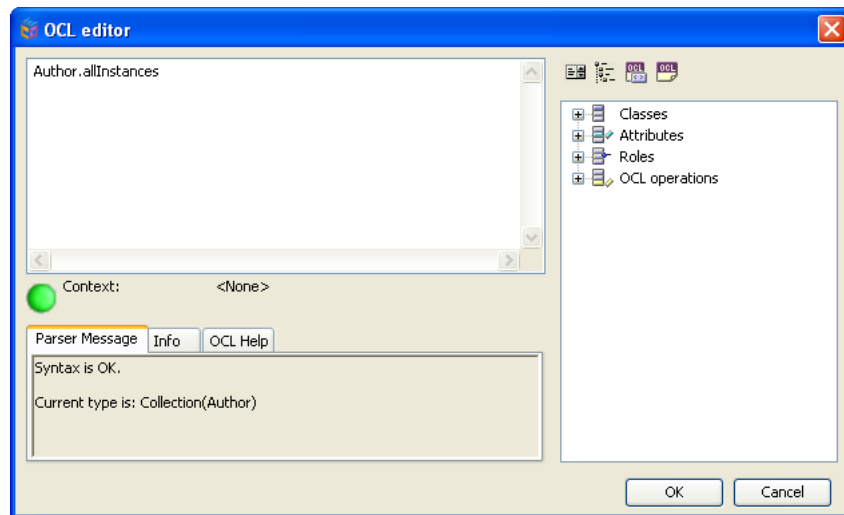
Property	Value
DataSource	ehArticleTypes
DisplayMember	Name
ValueMember	Self

Finally you would need to expand the DataBindings property on the combo box and set SelectedValue to ehAuthorArticles.ArticleType, the column that should be updated with the newly selected value.

# 15 Master detail example

No WinForms .NET DB based document would be complete without the customary "Master-Detail" example. This example will demonstrate how to display only articles for a selected author, and how new articles added are automatically associated with the selected author.

1. Create a new ECO Windows Form and name it MasterDetailForm.cs
2. Set the EcoSpaceTypeName on rhRoot.
3. Add an ExpressionHandle to the new form and name it ehAuthors.
4. Set it's RootHandle to "rhRoot".
5. Set its Expression to "Author.allInstances".



6. Set "AddDefaultProperties" to false.
7. Edit the columns for ehAuthors and add a new column
  1. Name = FullName
  2. Expression = self.FullName
  3. Repeat for Salutation, FirstName, and LastName.
8. Add a DataGridView to the form and name it "GridAuthors"
9. Set its DataSource "ehAuthors".

With the master grid set up it is time to move on to creating the detail grid.

10. Add a CurrencyManagerHandler to the form
  1. Name = cmhSelectedAuthor
  2. RootHandle = ehAuthors
  3. BindingContext = GridAuthors.

The CurrencyManagerHandle takes a collection of elements from its RootHandle (ehAuthors) and uses a single item from



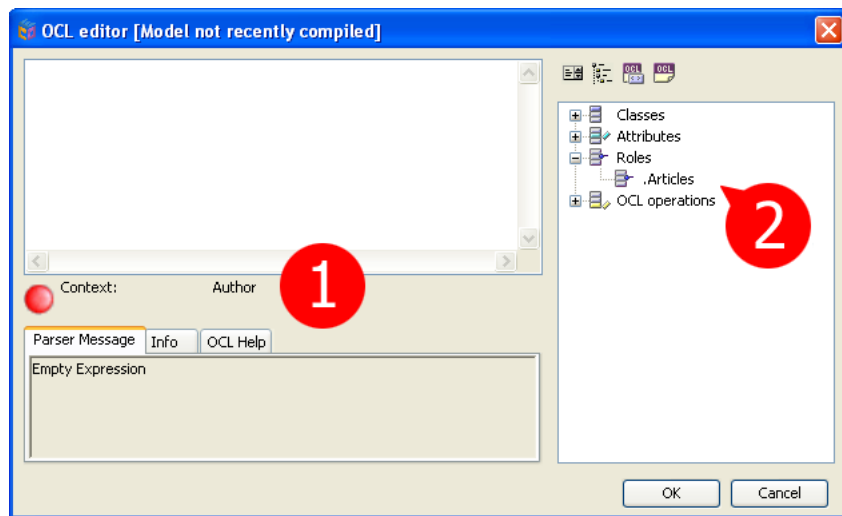
that collection as its own Element for data binding. To determine which element in the collection to use it asks its BindingContext (GridAuthors) which row number is selected. Whenever the selected row in the BindingContext changes the CurrencyManagerHandle will update its element property and cause all attached controls to rebind to the new value.

11. Add an ExpressionHandle to the form

1. Name = ehAuthorArticles
2. RootHandle = cmhSelectedAuthor

12. Now click the Expression [...] property editor.

When the OCL editor appears you will notice the following:



(1) The context for the expression is Author.

(2) Members of the Author class appear in the list on the right, including the Author's articles.

13. Set the expression to self.Articles and click OK.

14. Now add another DataGridView to the form.

- Name = GridAuthorArticles.
- DataSource = ehAuthorArticles
- Columns = Keep only Title, ArticleType, State, and ReasonRejected.

15. Now we need a button to add new Articles. Create a Button.

- Name = ButtonAddArticle
- "EcoListAction on EcoListActions" = Add
- "RootHandle on EcoListActions" = ehAuthorArticles
- Text = Add Article

16. Finally, add a button to the main form and add the following source code

```
private void ButtonMasterDetail_Click(object sender, EventArgs e)
{
    new MasterDetailForm(EcoSpace).ShowDialog();
}
```

Now run the application and click the Master Detail button on the main form. You are now able to perform master-detail

operations by creating and deleting authors, and also creating and deleting articles. Note that any new articles are automatically associated with the selected author.

MasterDetailForm

	FullName	Salutation	FirstName	LastName
▶	Mr Jon Smith	Mr	Jon	Smith
	Mr Fred Smith	Mr	Fred	Smith

	Title	ArticleType	ReviewState	ReasonRejected
▶	My second article	ArticleType		
	My first article	ArticleType	CheckingSpelling	

Add article

In addition to seeing the full name of the author remember that this is a derived settable member. You can replace the value by separating the elements of the name with a forward slash.

Mr/John/Smith Jones

	FullName	Salutation	FirstName	LastName
▶	Mr Jon Smith	Mr	Jon	Smith

	FullName	Salutation	FirstName	LastName
✎	Mr/Peter/Morris	Mr	Jon	Smith

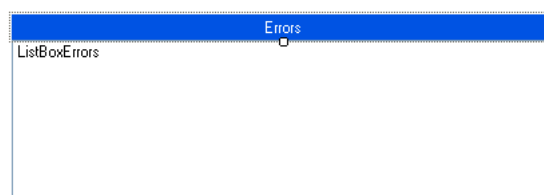
	FullName	Salutation	FirstName	LastName
	Mr Peter Morris	Mr	Peter	Morris

# 16 Validating input

During the modeling stage a number of OCL constraints were defined on the Author and Article classes. If you recall, the return type of these expressions were all Boolean. Evaluating these constraints allows the application to determine whether or not the object is valid as defined by its constraints.

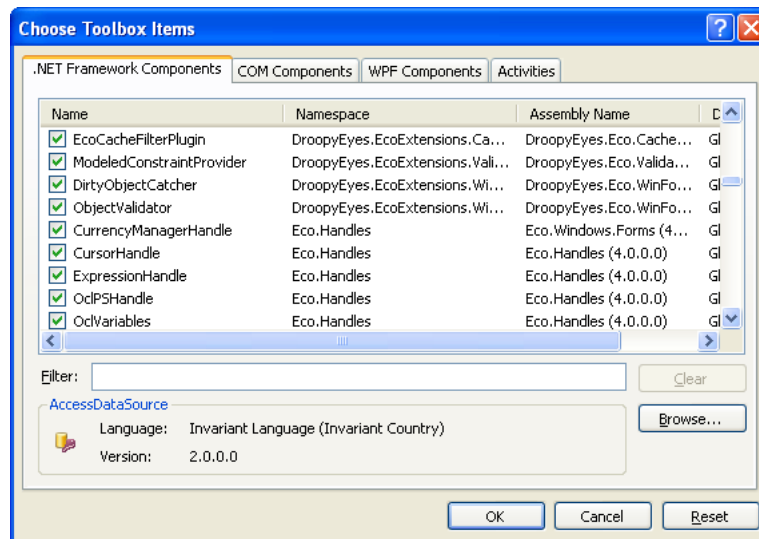
Next validation will be added to the application. This will prevent any invalid objects from being saved by preventing the user from saving changes to an article if it has any broken constraints. As validation is something which will be required throughout an application these steps will create a reusable control for providing validation and displaying the broken constraints.

1. Right-click the QuickStart.WinForms application.
2. From the context menu select Add->User Control.
3. Name the control MyEcoValidator.cs
4. Add a label to the control.
  1. Name = LabelErrors
  2. AutoSize = False
  3. Dock = Top
  4. BackColor = ActiveCaption
  5. ForeColor = ActiveCaptionText
  6. Text = "Errors"
  7. TextAlign = MiddleCenter
5. Add a ListBox to the form
  1. Name = ListBoxErrors
  2. Dock = Fill



6. Now add a ModeledConstraintProvider to the controller

Note: If you do not have an ECO Extensions section on your tool palette you can add one by right-clicking the palette and selected "Add tab". Then right-click again and select Choose Items. You can add the ECO Extensions DLLs by browsing to {Program Files}\CapableObjects\ECO\{VersionNumber}\Bin and selecting all DLLs starting with "DroopyEyes" in turn, note that not all DLLs expose components which may appear on the palette. Once the DLLs have been added you will need to tick each component to have it appear.



7. Add a DirtyObjectCatcher to the control.

8. Add an ObjectValidator to the control

1. ConstraintProvider = modeledConstraintProvider1

2. DirtyObjectCatcher = dirtyObjectCatcher1

The DirtyObjectCatcher will identify which objects were modified whilst the owning form has focus.

The ObjectValidator will validate these modified objects.

The ModeledConstraintProvider will provide constraint information to the ObjectValidator from the modeled constraints.

To complete the component's implementation it needs to be passed a reference to the EcoSpace the form uses. In addition it needs to be able to inform the parent form as to whether or not there are any broken constraints. The following code implements these features and should be added to your control's source.

```
public void Activate(EcoSpace ecoSpace)
{
    if (ecoSpace == null)
        throw new ArgumentNullException("ecoSpace");
    if (ParentForm == null)
        throw new NullReferenceException("ParentForm");

    dirtyObjectCatcher1.Activate(ParentForm, ecoSpace);
}

public bool ContainsChanges
{
    get { return dirtyObjectCatcher1.ContainsChanges; }
}

public bool HasErrors
{
    get { return ListBoxErrors.Items.Count > 0; }
}

public void UpdateDatabase()
{
    dirtyObjectCatcher1.UpdateDatabase();
}
```

You need to add "Eco.Handles" to the list of using-clauses.

To populate the `ListBoxErrors.Items` three events need to be implemented on the `ObjectValidator`:

```
private void objectValidator1_BeginValidation(object sender, EventArgs e)
{
    ListBoxErrors.BeginUpdate();
    ListBoxErrors.Items.Clear();
}

private void objectValidator1_ConstraintBroken(object sender, ConstraintBrokenEventArgs e)
{
    ListBoxErrors.Items.Add(e.Constraint.Name);
}

private void objectValidator1_EndValidation(object sender, EventArgs e)
{
    ListBoxErrors.EndUpdate();
}
```

### Adding the control to a form

Before adding the validator to a form a new form will be created for editing a single article. This will ensure that all errors you see are related to a single article, and will later illustrate an added benefit of using the `DirtyObjectCatcher`.

1. Add a new WinForm.
2. Name it "ArticleForm.cs".
3. Add a ReferenceHandle
  1. Name = `rhArticle`.
  2. `EcoSpaceTypeName = QuickStart.QuickStartEcoSpace`
  3. `StaticValueTypeName = Article`
4. Add the following property to the form's source.

```
public QuickStartEcoSpace EcoSpace
{
    get { return (QuickStartEcoSpace)rhArticle.EcoSpace; }
    set { rhArticle.EcoSpace = value; }
}
```

5. Change the form's constructor like so that it accepts a `QuickStartEcoSpace` and an `Article`.

```
public ArticleForm(QuickStartEcoSpace ecoSpace, Article article)
{
    if (ecoSpace == null)
        throw new ArgumentNullException("ecoSpace");
    if (article == null)
        throw new ArgumentNullException("article");

    InitializeComponent();
    EcoSpace = ecoSpace;
    rhArticle.SetElement(article);
}
```

6. Add some TextBox and Label controls in order to edit the article. Just Title, Summary, and Body will suffice for this example. You can copy the Textboxes from AuthorArticleForm and adjust the databinding properties to the new handle.
7. Compile the application so the IDE is aware of your new UserControl. It should now appear in the Toolbox in a tab called "QuickStart.WinForms Components". Drag the usercontrol from the toolbox to ArticleForm.
8. Switch back to the AuthorArticles form.
9. In the DoubleClick event for GridAuthorArticles enter the following code, this will show the new ArticleForm for the article that was double-clicked.

```
private void GridAuthorArticles_DoubleClick(object sender, EventArgs e)
{
    //Get the selected element
    IElement articleElement = CurrencyManagerHandle.CurrentElement(GridAuthorArticles);
    if (articleElement != null)
    {
        //Get the article it refers to
        Article article = articleElement.GetValue<Article>();

        //Create the article form and show it
        ArticleForm articleForm = new ArticleForm(EcoSpace, article);
        using (articleForm)
            articleForm.ShowDialog();
    }
}
```

9. Run the application.
10. Click the "Show articles" button.
11. Double-click an article in the list. You will now see the form appear.

Notice that you can change the data so that it is invalid, for example by having no Title set. When you close the form the changes are still present on the AuthorArticlesForm. Next validation will be added to the form so that changes cannot be committed if there are any broken constraints.

12. Look in the Tool Box and you should see MyEcoValidator listed as the first item. Drag this and drop it onto the ArticleForm.

13. Add a button to the form

1. Name = SaveChangesButton
2. Text = "Save"
3. In the Click event add the following code

```
private void SaveChangesButton_Click(object sender, EventArgs e)
{
    if (myEcoValidator1.HasErrors)
        MessageBox.Show("Please correct your errors");
    else
    {
        myEcoValidator1.UpdateDatabase();
        Close();
    }
}
```

14. Finally in the Form.Load event add the following code to activate the DirtyObjectCatcher component within MyEcoValidator.

```
private void ArticleForm_Load(object sender, EventArgs e)
{
    myEcoValidator1.Activate(EcoSpace);
}
```

15. Run the application again.

16. Click "Show Articles".

17. Double-click an article in the list.

18. Now make the article invalid by clearing out the Title and trying clicking the Save button.

The screenshot shows a Windows Forms application titled "ArticleForm". It contains three text controls: "Title", "Description", and "Body". The "Title" control has a validation error message displayed below it: "Title must be at least 5 characters long". A modal dialog box is open in the center of the form, titled "Please correct your errors", with an "OK" button. The "Description" control contains the text "My article is written in Latin." and the "Body" control contains a block of Lorem Ipsum text. At the bottom of the form is a "Save" button.

You will notice that the list of errors update automatically as you alter the data and change focus between the controls. For example, you may alter Title but the error message will not appear until the `TextBoxTitle` loses focus, this is because WinForms data binding applies its changes when the control no longer has focus.

As you edit the Title for the article and then change focus to another control you may observe the article title in the `AuthorArticlesForm` beneath it also changing. However, when you click the Close button for the `ArticleForm` you will see that the article title in the `AuthorArticlesForm` is restored to its original value. This is a bonus of using the `DirtyObjectCatcher` inside the `MyEcoValidator` control.

The `DirtyObjectCatcher` not only catches objects which have been modified whilst its parent form is active, it also records all of these changes in an ECO undo block. When you call `DirtyObjectCatcher.UpdateDatabase()` only the changes made within the current form are saved to the data storage, but if you close the form these changes will be reversed and the local ECO cache will be restored to its original state.



# 17 Setting maximum input lengths

Many UI controls have a property which allows you to limit the length of user input. Rather than having to set these values manually ECO has a feature which can perform this task automatically.

1. Open the ArticleForm.
2. Add the following line of code to the bottom of the constructor.

```
Eco.Handles.MaxLengthSupportFunctions.UpdateMaxLengths(Controls);
```

3. Run the application.
4. Bring up the ArticleForm.
5. Try to enter more than 64 characters into the TextBoxTitle control.

When you first call UpdateMaxLengths ECO will check each loaded assembly to see if it has the reflection attribute [MaxLengthSupportLibrary] applied. When such an assembly is identified each class within the assembly is reflected over to see if the reflection attribute [MaxLengthSupportProvider] has been applied to it. If the class does have this reflection attribute applied then an instance of the class will be created and added to a dictionary.

Note: You may manually register assemblies with MaxLengthSupportProviders in by executing RegisterMaxLengthSupportLibrary(Assembly), or register individual support providers by executing RegisterMaxLengthSupportProvider(ICollection<IMaxLengthSupportProvider>, Type).

The provider for TextBox can be seen in the following example

```
//The class is decorated with the MaxLengthSupportProvider reflection attribute, the
//control type is also specified.
[MaxLengthSupportProvider(typeof(TextBox))]

//The class implements IMaxLengthSupportProvider
public class TextBoxMaxLengthSupportProvider : IMaxLengthSupportProvider
{
    public TextBoxMaxLengthSupportProvider()
    {
    }

    //SetMaxLength is executed whenever a control of a relevant type is encountered
    public void SetMaxLength(object obj)
    {
        if (obj == null)
            throw new ArgumentNullException("obj");
        TextBox textBox = (TextBox)obj;

        //Get the binding object applied to the Text property
        Binding binding = textBox.DataBindings["Text"];

        //Only continue if bound to an ECO handle
        if (binding != null && binding.DataSource is ElementHandle)
        {
            ElementHandle handle = binding.DataSource as ElementHandle;
            PropertyDescriptorCollection pdc = (handle as
```

```
ITypedList).GetItemProperties(null);
    RenderedTuplePropertyDescriptor prop =
pdc.Find(binding.BindingMemberInfo.BindingField, true) as
RenderedTuplePropertyDescriptor;
    textBox.ReadOnly = textBox.ReadOnly || prop.IsReadOnly;
    textBox.MaxLength = MaxLengthSupportFunctions.GetMaxLength(prop, handle,
textBox.MaxLength);
    }
}
```

By copying the majority of this demo source code it is possible to implement `MaxLengthSupportProvider` classes for various 3rd party controls such as DevExpress XtraGrid etc. In fact, as the `MaxLengthSupportFunctions` class is in an ECO assembly that is not tied to any particular UI it is also possible to create `MaxLengthSupportProvider` classes for other user interfaces such as ASP .NET.

# 18 Model driven drag 'n' drop

Having full model information available at runtime enables you to extend the way your application works in many ways. One example of this is the automatic drag and drop support provided as a standard feature in ECO.

1. Add a button to the application's main form.

1. Name = "ButtonDragDrop"
2. Text = "Drag drop"

In its Click event add the following code.

```
new DragDropForm(EcoSpace).ShowDialog();
```

2. Create a new WinForm named DragDropForm.cs

3. Add a ReferenceHandle to the form

1. Name = "rhRoot"
2. EcoSpaceTypeName = "QuickStart.QuickStartEcoSpace"

4. Optionally add an EcoSpace property.

```
public QuickStartEcoSpace EcoSpace
{
    get { return (QuickStartEcoSpace)rhRoot.EcoSpace; }
}
```

5. Add a QuickStartEcoSpace parameter to the constructor and set rhRoot.EcoSpace to its value.

```
public DragDropForm(QuickStartEcoSpace ecoSpace)
{
    if (ecoSpace == null)
        throw new ArgumentNullException("ecoSpace");

    InitializeComponent();
    rhRoot.EcoSpace = ecoSpace;
}
```

With the basic ECO form configuration complete it is now time to add some controls with automatic drag drop enabled.

6. Add an ExpressionHandle to the form

1. Name = "ehArticleTypes"
2. RootHandle = rhRoot
3. Expression = ArticleType.allInstances

7. Add a DataGridView to the form

1. Name = "GridViewArticleTypes"
2. DataSource = ehArticles

As demonstrated earlier a CurrencyManagerHandle will be added which will use the currently selected ArticleType as the context for its child handles.

8. Add a CurrencyManagerHandle to the form

1. Name = "cmhSelectedArticleType"
2. RootHandle = ehArticleTypes
3. BindingContext = GridViewArticleTypes

Next a list of all articles of the selected ArticleType will be displayed.

9. Add an ExpressionHandle to the form

1. Name = "ehArticleTypeArticles"
2. RootHandle = cmhSelectedArticleType
3. Expression = self.Articles

10. Add a DataGridView to the form

1. Name = "GridViewArticleTypeArticles"
2. DataSource = ehArticleTypeArticles

Note: "self" is case sensitive and should be lowercase.

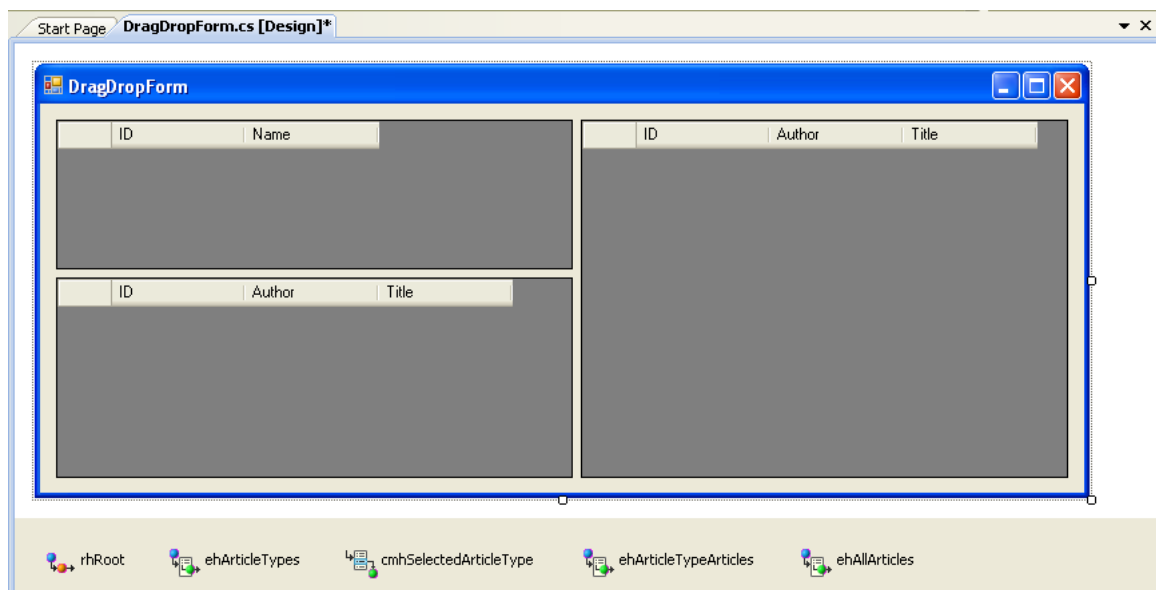
Run the application and click ButtonDragDrop. As you select each article type in the first grid you will see a list of articles associated with the selected ArticleType. So far this is the same as the Master Detail example. Next a list of all articles will be presented and the grids will be configured to allow the user to drag an article from the complete list and drop it into the list for the selected ArticleType.

11. Add an ExpressionHandle to the form

1. Name = ehAllArticles
2. RootHandle = rhRoot
3. Expression = Article.allInstances

12. Add a DataGridView to the form

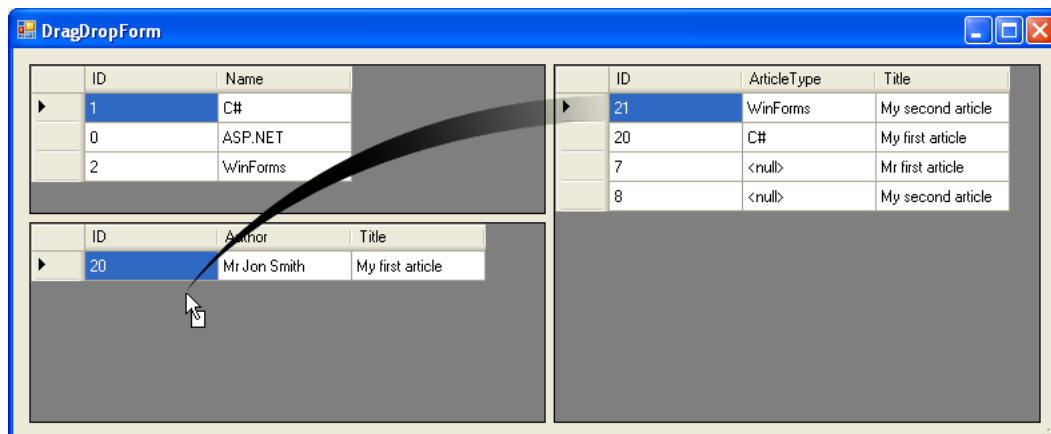
1. Name = "GridViewAllArticles"
2. DataSource = ehAllArticles



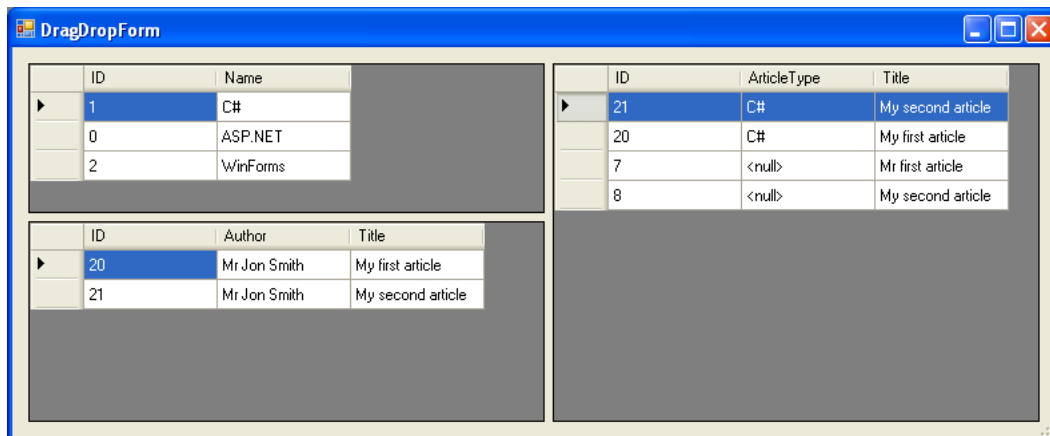
Next the easy part, adding drag and drop support!

13. Add an EcoDragDropExtender component to the form.
  1. Name = EcoDragDrop
14. Single-click GridViewAllArticles (on the right).
15. Set its "EcoDragSource on EcoDragDrop" property to True.
16. Single-click GridViewArticleTypeArticles (the one on the bottom left).
17. Set its "EcoDropTarget on EcoDragDrop" property to True.

Now run the application and show the Drag Drop form.



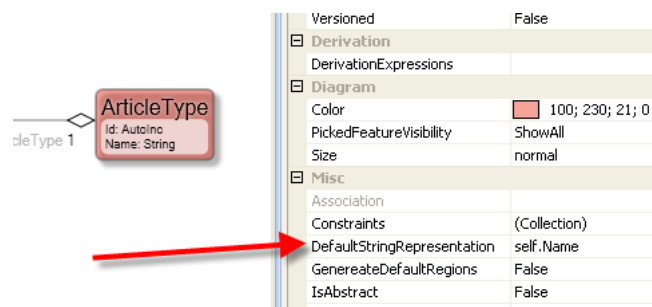
Click the blank column to the left of the GridViewAllArticles, hold the mouse button down, then drag it and drop it onto the GridViewArticleTypeArticles grid. When you release the mouse you will see that the dragged article now becomes part of the selected ArticleType's Article list.



The EcoDragDropExtender has initiated a drag drop operation. As the mouse passes over a DropTarget ECO will inspect the control to see how it is bound to its data source. If the data source is an ECO handle the extender will check if its element is a domain element such as SomeObject.SomeAssociation. If it is then the extender will inspect the model information to see if the dragged object instance (Article) is compatible with the target element, which in this case is a Collection(Article). When you release the mouse the dragged object is added to the target element.

The result is that the dragged article is added to the ArticleType.Articles, and consequently Article.ArticleType is set to the select ArticleType. As you can see in the two preceding screenshots, the ArticleType has changed from WinForms to C#.

Note: The default column added for an association to a single object is self.AssociationName.AsString - AsString will use the OCL expression in the class's Default String Representation to render the class as a string, and by default this returns the name of the class. In this example changed the Default String Representation for ArticleType to "Name".



# 19 Summary

This article has demonstrated how to use a single ECO class package to form a business model for a WinForms application. Instances of the contained business classes were then created and modified using standard controls bound to ECO handle components. Using this approach it is possible to use the classic "data" approach for writing applications but with additional benefits such as inheritance, virtual methods, and ECO state machines.

The examples covered typical scenarios such as master-detail relationships, specifying links using a ComboBox, validating user input, and restricting the length of user input. Finally a form was created to demonstrate automatic drag and drop, the kind of advanced application support that may be added to any application by having rich model information available at runtime.

## Index

### A

Adding more controls to the articles form 21  
Associating objects using a ComboBox 33

### C

Creating a custom column 29  
Creating a secondary form for editing articles 17  
Creating and modifying objects 14  
Creating the application 3

### E

ECO data binding 9  
ECO handles 9

### H

How models are constructed 8

### L

Limiting input lengths 22

### M

Master detail example 37  
Model driven drag 'n' drop 48

### O

Observing state machine behavior 24  
Overview 1

### P

Prerequisites and goals 2

### R

Rejecting and Editing articles 26  
Running the application for the first time 16

### S

Setting maximum input lengths 46

Summary 52

### U

Understanding the WinForms Project 4  
Using multiple EcoSpace instances 30

### V

Validating input 40